Silvia Rossi

Agent as Intentional Systems

Lezione n. 2

Corso di Laurea:
Informatica

Insegnamento:
Sistemi
multi-agente

Email:
silrossi@unina.it

A.A. 2014-2015

Agenti e Ambienti

(RN, WS)

Environments – *Accessible* vs. *inaccessible*

- An accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state

- Most moderately complex environments (including, for example, the everyday physical world and the Internet) are inaccessible
- The more accessible an environment is, the simpler it is to build agents to operate in it

- Quality of information → quality of action decisions
- Environment more accessible → agent easier to construct

Environments – *Deterministic* vs. *non-deterministic*

- A deterministic environment is one in which any action has a single guaranteed effect — there is no uncertainty about the state that will result from performing an action
- The physical world can to all intents and purposes be regarded as non-deterministic

- Agents have limited spheres of influence and limited sensoric capabilities

  → no complete control

  → non-determinism (from an individual agent's point of view) even in "overall deterministic" environments

- Non-determinism → Actions can fail

Environments - *Episodic* vs. *non-episodic*

- In an episodic environment, the performance of an agent is dependent on a number of discrete episodes, with no link between the performance of an agent in different scenarios
- Episodic environments are simpler from the agent developer's perspective because the agent can decide what action to perform based only on the current episode — it need not reason about the interactions between this and future episodes

Environments - *Static* vs. *dynamic*

- A static environment is one that can be assumed to remain unchanged except by the performance of actions by the agent
  - Predictable. Information gathering:
    Once → preconditions established → plan → execution
- A dynamic environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control
  - Unpredictable → If precond. ϕ for action α
    holds at time t0 → no guarantee for ϕ at time t1 → need for constant information gathering → action plans can fail because preconditions change over time

Environments – *Discrete* vs. *continuous*

- An environment is discrete if there are a fixed, finite number of actions and percepts in it
- Russell and Norvig give a chess game as an example of a discrete environment, and taxi driving as an example of a continuous one
- Continuous environments have a certain level of mismatch with computer systems
- Discrete environments could *in principle* be handled by a kind of “lookup table”

Environment types

	Chess with a clock	Chess without a clock	Taxi driving
Fully observable	Yes	Yes	No
Deterministic	Strategic	Strategic	No
Episodic	No	No	No
Static	Semi	Yes	No
Discrete	Yes	Yes	No
Single agent	No	No	No

- The environment type largely determines the agent design
- The real world is (of course) partially observable, stochastic, sequential, dynamic, continuous, multi-agent

Agenti Come Sistemi Intenzionali

(W)

Agents as Intentional Systems

- When explaining human activity, it is often useful to make statements such as the following:

  Janine took her umbrella because she *believed* it was going to rain.

  Michael worked hard because he *wanted* to possess a PhD.

- These statements make use of a *folk psychology*, by which human behavior is predicted and explained through the attribution of *attitudes*, such as believing and wanting (as in the above examples), hoping, fearing, and so on

- The attitudes employed in such folk psychological descriptions are called the *intentional* notions

Agents as Intentional Systems

- The philosopher Daniel Dennett coined the term *intentional system* to describe entities ‘whose behavior can be predicted by the method of attributing belief, desires and rational acumen’

- Dennett identifies different ‘grades’ of intentional system: ‘A *first-order* intentional system has beliefs and desires (etc.) but no beliefs and desires *about* beliefs and desires. …A *second-order* intentional system is more sophisticated; it has beliefs and desires (and no doubt other intentional states) about beliefs and desires (and other intentional states) — both those of others and its own’

- Is it legitimate or useful to attribute beliefs, desires, and so on, to computer systems?

- McCarthy argued that there are occasions when the *intentional stance* is appropriate:

> 'To ascribe *beliefs*, *free will*, *intentions*, *consciousness*, *abilities*, or *wants* to a machine is *legitimate* when such an ascription expresses the same information about the machine that it expresses about a person. It is *useful* when the ascription helps us understand the structure of the machine, its past or future behavior, or how to repair or improve it. It is perhaps never *logically required* even for humans, but expressing reasonably briefly what is actually known about the state of the machine in a particular situation may require mental qualities or qualities isomorphic to them. Theories of belief, knowledge and wanting can be constructed for machines in a simpler setting than for humans, and later applied to humans. Ascription of mental qualities is *most straightforward* for machines of known structure such as thermostats and computer operating systems, but is *most useful* when applied to entities whose structure is incompletely known'.

- What objects can be described by the intentional stance?
- As it turns out, more or less anything can. . . consider a light switch:

> ‘It is perfectly coherent to treat a light switch as a (very cooperative) agent with the capability of transmitting current at will, who invariably transmits current when it believes that we want it transmitted and not otherwise; flicking the switch is simply our way of communicating our desires’. (Yoav Shoham)

- But most adults would find such a description absurd!
  Why is this?

Agents as Intentional Systems

- The answer seems to be that while the intentional stance description is consistent,

  > . . . it does not *buy us anything*, since we essentially understand the mechanism sufficiently to have a simpler, mechanistic description of its behavior.
  >
  > (Yoav Shoham)

- Put crudely, the more we know about a system, the less we need to rely on animistic, intentional explanations of its behavior
- But with very complex systems, a mechanistic, explanation of its behavior may not be practicable
- *As computer systems become ever more complex, we need more powerful abstractions and metaphors to explain their operation — low level explanations become impractical. The intentional stance is such an abstraction*

Agents as Intentional Systems

- The intentional notions are thus *abstraction tools*, which provide us with a convenient and familiar way of describing, explaining, and predicting the behavior of complex systems
- Remember: most important developments in computing are based on new *abstractions*:
  - procedural abstraction
  - abstract data types
  - objects

  Agents, and agents as intentional systems, represent a further, and increasingly powerful abstraction
- So agent theorists start from the (strong) view of agents as intentional systems: one whose simplest consistent description requires the intentional stance

- This *intentional stance* is an *abstraction tool* — a convenient way of talking about complex systems, which allows us to predict and explain their behavior without having to understand how the mechanism actually works
- Now, much of computer science is concerned with looking for abstraction mechanisms (witness procedural abstraction, ADTs, objects,…)

  So why not use the intentional stance as an abstraction tool in computing — to explain, understand, and, crucially, program *computer systems?*
- This is an important argument in favor of agents

- Other 3 points in favor of this idea:
- Characterizing Agents:
  - It provides us with a familiar, non-technical way of *understanding & explaining* agents
- Nested Representations:
  - It gives us the potential to specify systems that *include representations of other systems*
  - It is widely accepted that such nested representations are essential for agents that must cooperate with other agents

- Post-Declarative Systems:
  - This view of agents leads to a kind of post-declarative programming:
    - In procedural programming, we say exactly *what* a system should do
    - In declarative programming, we state something that we want to achieve, give the system general info about the relationships between objects, and let a built-in control mechanism (e.g., goal-directed theorem proving) figure out what to do
    - With agents, we give a very abstract specification of the system, and let the control mechanism figure out what to do, knowing that it will act in accordance with some built-in theory of agency (e.g., the well-known Cohen-Levesque model of intention)

An aside…

- We find that researchers from a more mainstream computing discipline have adopted a similar set of ideas…
- In distributed systems theory, *logics of knowledge* are used in the development of *knowledge based protocols*
- The rationale is that when constructing protocols, one often encounters reasoning such as the following:

  IF process *i* knows process *j* has received message m_1
  THEN process *i* should send process *j* the message m_2

- In DS theory, knowledge is *grounded* — given a precise interpretation in terms of the states of a process; we'll examine this point in detail later

The Intentional Stance

- Intentional Stance: Attributing attitudes (beliefs, desires, whishes) to systems whose precise internal function is unknown. (Controversial example: Light switch)
- Physical Stance: Observe → analyze function principles (induce general description) → predict future behavior (through deduction). (Example: Apple and Newton‘s second law)
- Design Stance: Use knowledge about design intention of an object to predict behavior (Example: Alarm Clock)

Most real world systems too complex for physical or design stance. → Why not use intentional stance as means of complexity reduction (Compare Obj.Orient.)

Architetture per Agenti

(W, RN)

Abstract Architecture for Agents

- Assume the environment may be in any of a finite set *E* of discrete, instantaneous states:

$$E = \{e, e', \ldots\}.$$

- Agents are assumed to have a repertoire of possible actions available to them, which transform the state of the environment:

$$Ac = \{\alpha, \alpha', \ldots\}$$

- A *run*, *r*, of an agent in an environment is a sequence of interleaved environment states and actions:

$$r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_{u-1}} e_u$$

- Let:
  - R be the set of all such possible finite sequences (over *E* and *Ac*)
  - R^{Ac} be the subset of these that end with an action
  - R^{E} be the subset of these that end with an environment state

State Transformer Functions

- A *state transformer* function represents behavior of the environment: $\tau : \mathcal{R}^{Ac} \rightarrow \wp(E)$

- Note that environments are…
  - *history dependent*
  - *non-deterministic*
- If $\tau(r)=\varnothing$, then there are no possible successor states to r. In this case, we say that the system has *ended* its run
- Formally, we say an environment Env is a triple $Env = \langle E, e_0, \tau \rangle$ where: E is a set of environment states, $e_0 \in E$ is the initial state, and τ is a state transformer function

- Agent is a function which maps runs to actions:

$$Ag : \mathcal{R}^E \rightarrow Ac$$

An agent makes a decision about what action to perform based on the history of the system that it has witnessed to date. Let AG be the set of all agents

Systems

- A *system* is a pair containing an agent and an environment
- Any system will have associated with it a set of possible runs; we denote the set of runs of agent Ag in environment Env by $R(Ag, Env)$
- (We assume $R(Ag, Env)$ contains only *terminated* runs)

- Formally, a sequence

$$(e_0, \alpha_0, e_1, \alpha_1, e_2, \ldots)$$

represents a run of an agent Ag in environment $Env = \langle E, e_0, \tau \rangle$ if:

1. e_0 is the initial state of Env
2. $\alpha_0 = Ag(e_0)$; and
3. For $u > 0$,

$$e_u \in \tau((e_0, \alpha_0, \ldots, \alpha_{u-1})) \quad \text{where}$$
$$\alpha_u = Ag((e_0, \alpha_0, \ldots, e_u))$$

Agent functions and programs

- Given an agent architecture
- An agent is completely specified by the agent function mapping percept sequences to actions
- One agent function (or a small equivalence class) is rational

- Aim: find a way to implement the rational agent function concisely

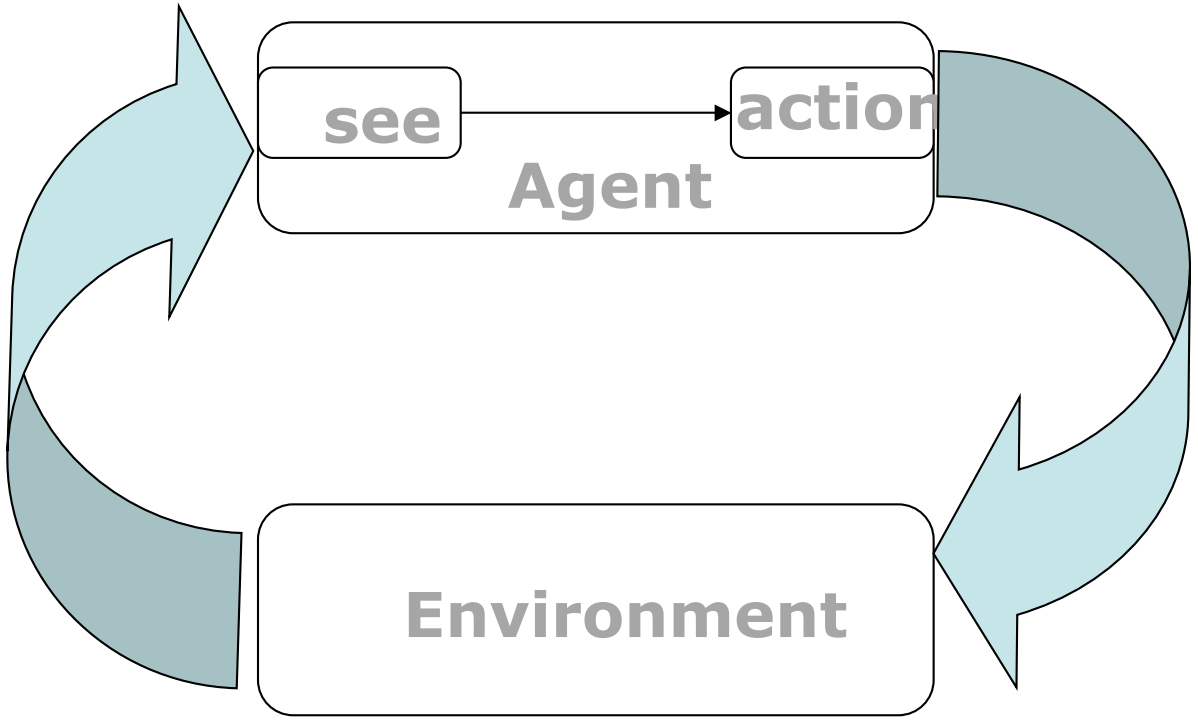
- Some agents decide what to do without reference to their history — they base their decision making entirely on the present, with no reference at all to the past
- We call such agents *purely reactive*:

$$action : E \rightarrow Ac$$

- A thermostat is a purely reactive agent

$$action(e) = \begin{cases} \text{off} & \text{if } e = \text{temperature OK} \\ \text{on} & \text{otherwise.} \end{cases}$$

- Now introduce *perception* system:



- The see function is the agent's ability to observe its environment, whereas the $action$ function represents the agent's decision making process
- *Output* of the see function is a *percept*:

$$see : E \rightarrow Per$$

which maps environment states to percepts, and $action$ is now a function

$$action : Per^* \rightarrow A$$

which maps sequences of percepts to actions

Table-lookup agent

- ```
  function TABLE-DRIVEN-AGENT(percept) returns action
    static: percepts, a sequence, initially empty
            table, a table, indexed by percept sequences, initially fully specified

    append percept to the end of percepts
    action ← LOOKUP(percepts, table)
    return action
  ```

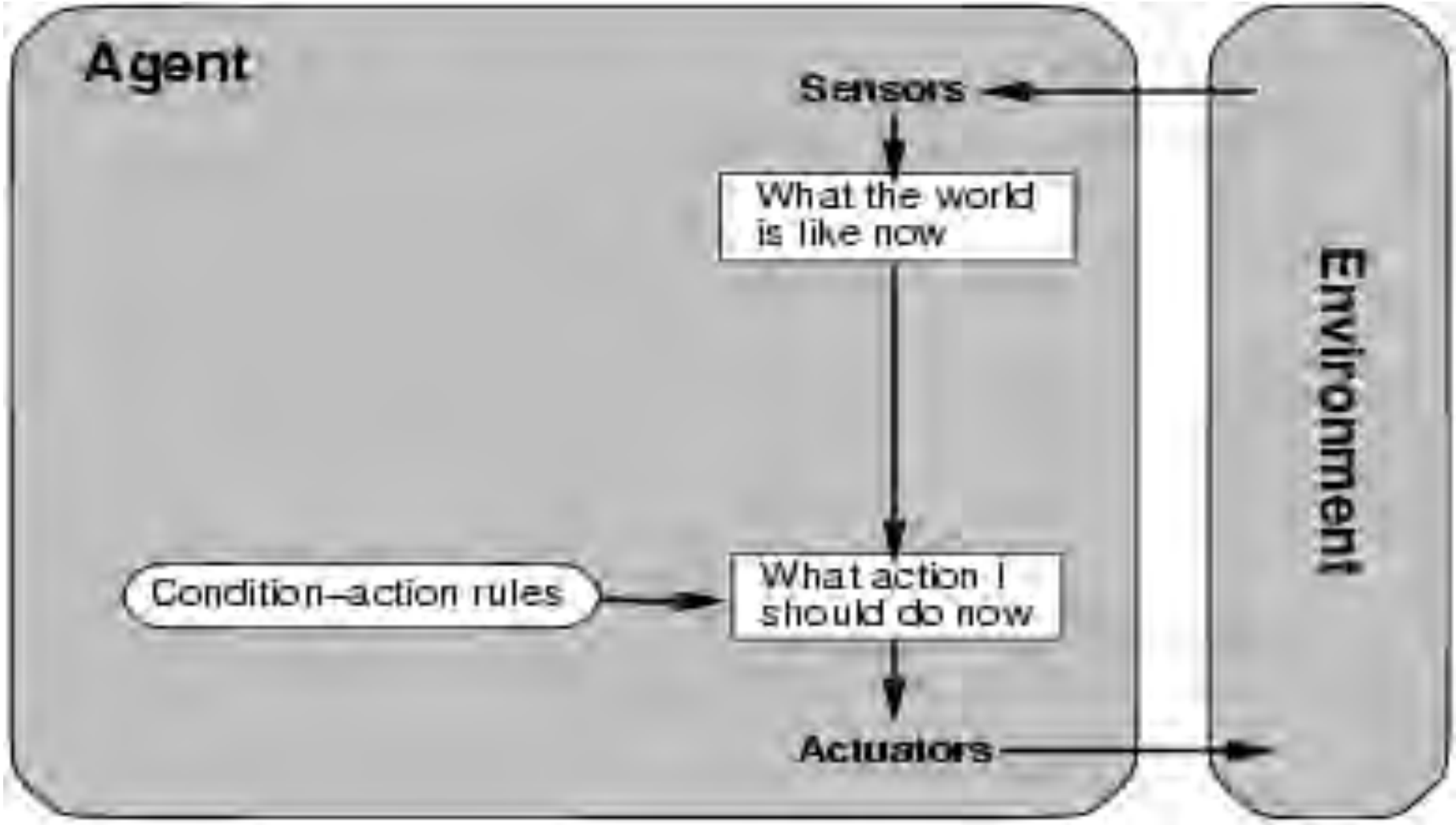
- Drawbacks:
  - Huge table (es. Chess 35^100 entry)
  - Take a long time to build the table
  - No autonomy
  - Even with learning, need a long time to learn the table entries
  ```

Agent types

- Four basic types in order of increasing generality:

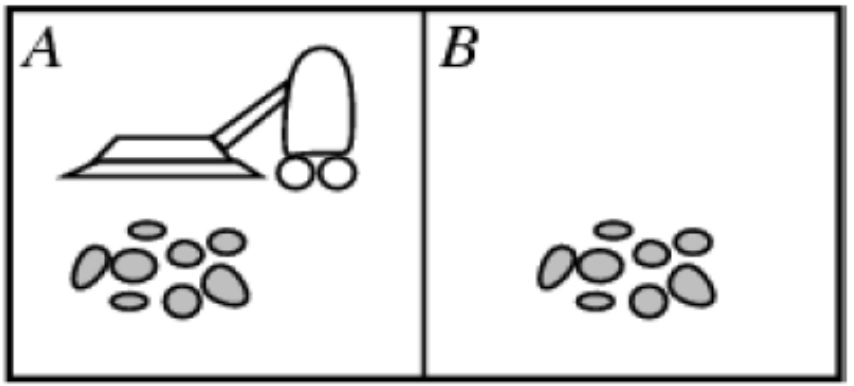
- Simple reflex agents
- Model-based reflex agents
- Goal-based agents
- Utility-based agents

Simple reflex agents



Simple reflex agents

- **Vacuum-cleaner world**



- **function *REFLEX_AGENT([location,status]): returns an action***

```
if status = Dirty then return Suck
    else if location = A then return Right
        else if location = B then return Left
```

Simple reflex agents

function SIMPLE-REFLEX-AGENT(PERCEPT): returns an action

static: *rules, a set of condition-action rules*

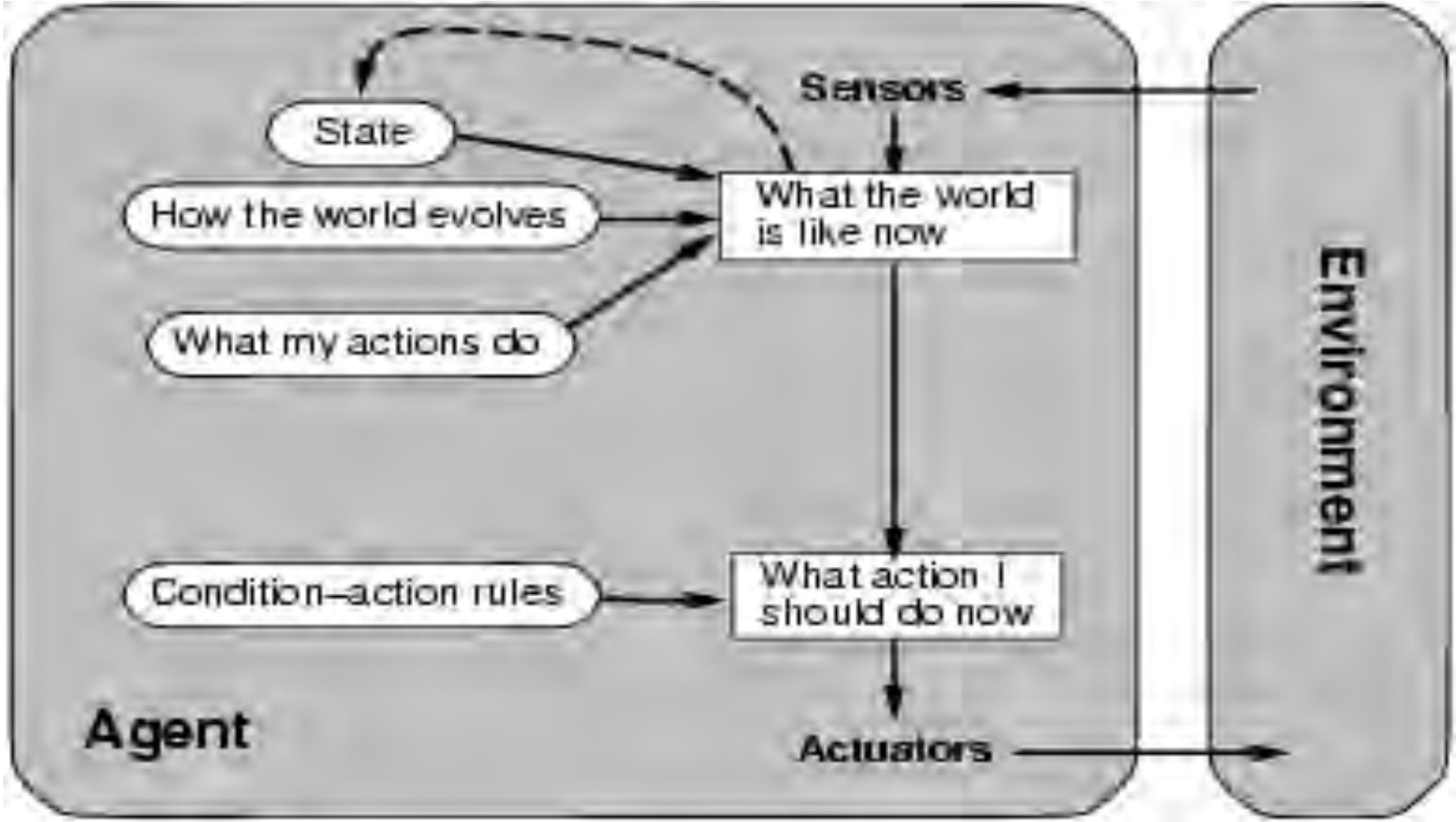
state <- INTERPRET-INPUT(percept)
rule <- RULE-MATCH(state,rules)
action <- RULE-ACTION[rule]
return *action*

function SIMPLE-REFLEX-AGENT(*percept*) **returns** *action*
static: *rules*, a set of condition-action rules

state ← INTERPRET-INPUT(*percept*)
rule ← RULE-MATCH(*state*, *rules*)
action ← RULE-ACTION[*rule*]
return *action*

Model-based reflex agents

Model of the world, prediction, evolution of the internal state



Internal data structure used to record information about the environment state and history.

Let I be the set of all internal states of the agent.

The perception function is:

$$see : E \rightarrow Per$$

A function $next$ maps an internal state and percept to an internal state:

$$next : I \times Per \rightarrow I$$

The action-selection function $action$ is a mapping

$$action : I \rightarrow Ac$$

from internal states to actions.

Agent Control Loop

1. Agent starts in some initial internal state i_0
2. Observes its environment state e, and generates a percept $see(e)$
3. Internal state of the agent is then updated via $next$ function, becoming $next(i_0, see(e))$
4. The action selected by the agent is $action(next(i_0, see(e)))$
5. Goto 2

function REFLEX-AGENT-WITH-STATE(*percept*) **returns** *action*
  static: *state*, a description of the current world state
    rules, a set of condition-action rules

  state ← UPDATE-STATE(*state*, *percept*)
  rule — RULE-MATCH(*state*, *rules*)
  action — RULE-ACTION[*rule*]
  state ← UPDATE-STATE(*state*, *action*)
  return *action*

Tasks for Agents

- We build agents in order to carry out *tasks* for us
- The task must be *specified* by us…
- But we want to tell agents what to do *without* telling them how to do it

Two most common types of tasks are *achievement tasks* and *maintenance tasks*:

1. *Achievement tasks* are those of the form "achieve state of affairs ϕ"
2. *Maintenance tasks* are those of the form "maintain state of affairs ψ"

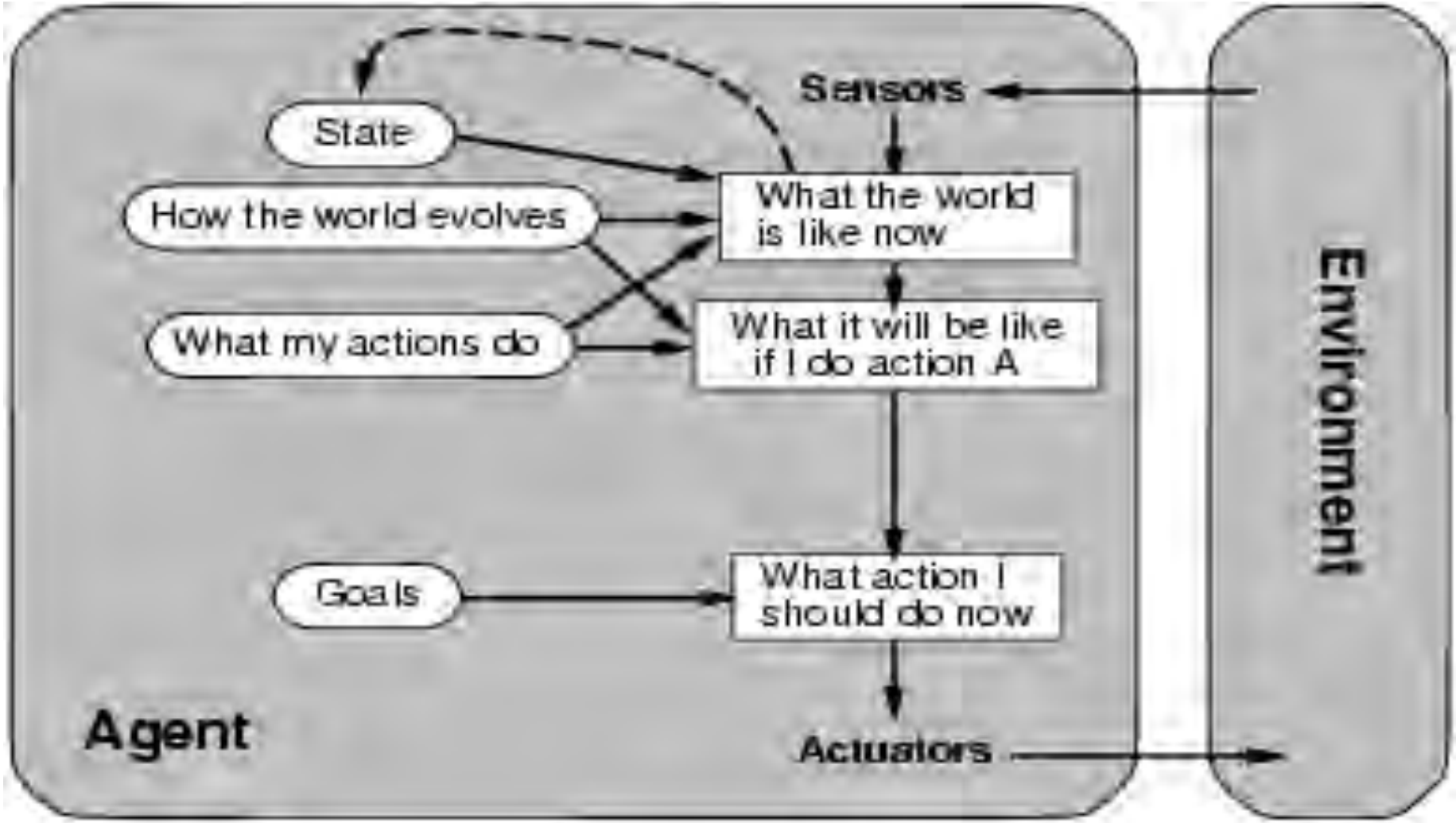
An achievement task is specified by a set G of "good" or "goal" states: $G \subseteq E$

The agent succeeds if it is guaranteed to bring about at least one of these states (we do not care which one — they are all considered equally good).

A maintenance goal is specified by a set B of "bad" states: $B \subseteq E$

The agent succeeds in a particular environment if it manages to *avoid* all states in B — if it never performs actions which result in any state in B occurring

Goal-based agents



Goal-based Agents

- No state-action rules, but goals
- Proactivity vs. reactivity
- To achieve the goals:
  - Generate sequence of actions
  - Search, Planning, Reasoning
  - Sense-Plan-Act

Utility Functions over States

- One possibility: associate *utilities* with individual states — the task of the agent is then to bring about states that maximize utility
- A task specification is a function

$$u : E \rightarrow R$$

which associates a real number with every environment state

- But what is the value of a *run*…
  - minimum utility of state on run?
  - maximum utility of state on run?
  - sum of utilities of states on run?
  - average?
- Disadvantage: difficult to specify a *long term* view when assigning utilities to individual states
  (One possibility: a *discount* for states later on.)

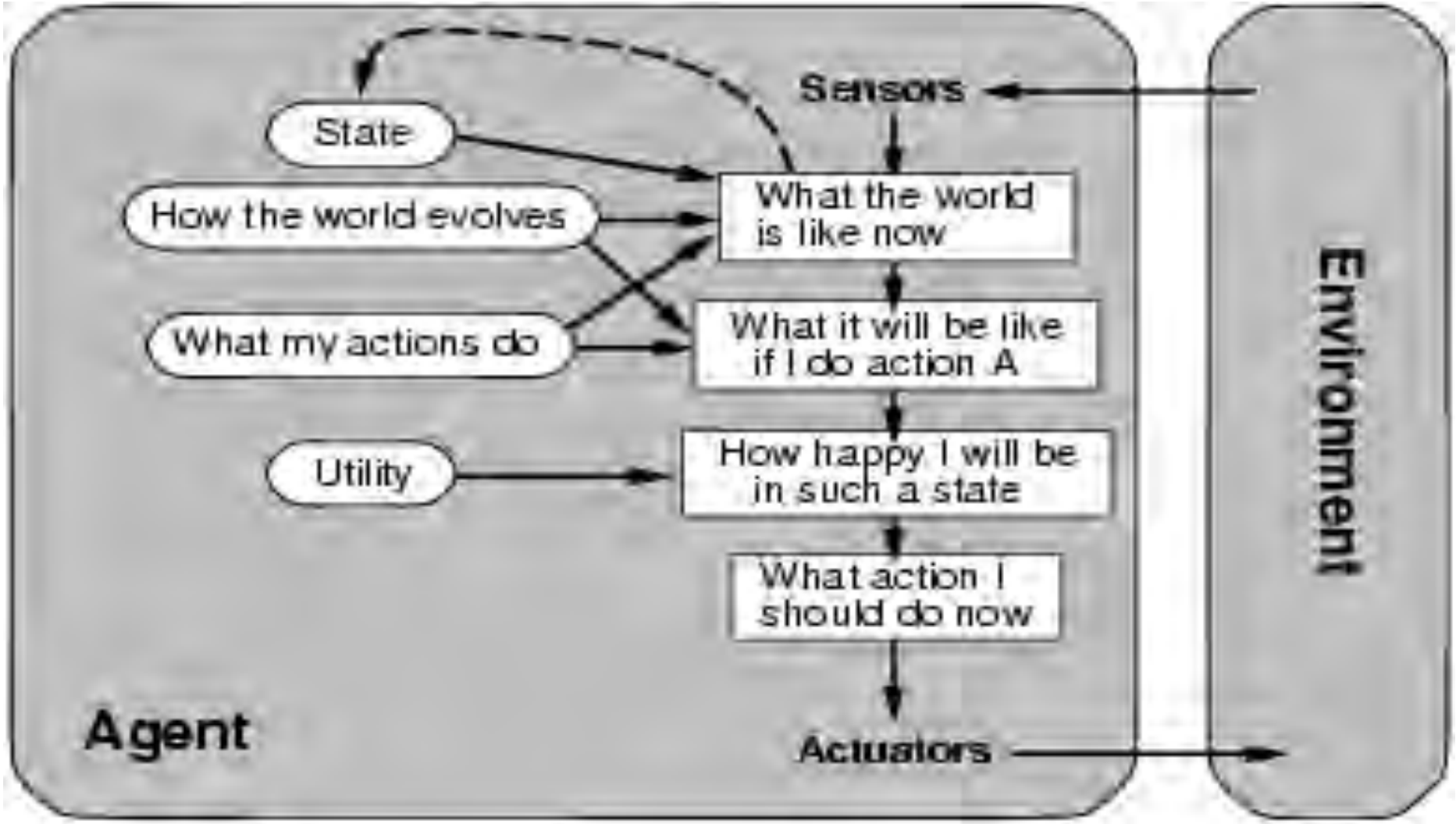
Utilities over Runs

- Another possibility: assigns a utility not to individual states, but to runs themselves:

$$u : \mathrm{R} \rightarrow R$$

- Such an approach takes an inherently *long term* view
- Other variations: incorporate probabilities of different states emerging
- Difficulties with utility-based approaches:
  - where do the numbers come from?
  - we don't think in terms of utilities!
  - hard to formulate tasks in these terms

Utility-based agents



Utility in the Tileworld

- Simulated two dimensional grid environment on which there are agents, tiles, obstacles, and holes
- An agent can move in four directions, up, down, left, or right, and if it is located next to a tile, it can push it
- Holes have to be filled up with tiles by the agent. An agent scores points by filling holes with tiles, with the aim being to fill as many holes as possible
- TILEWORLD changes with the random appearance and disappearance of holes
- Utility function defined as follows:

$$u(r) \hat{=} \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r}$$

- From Goldman and Rosenschein, AAAI-94:

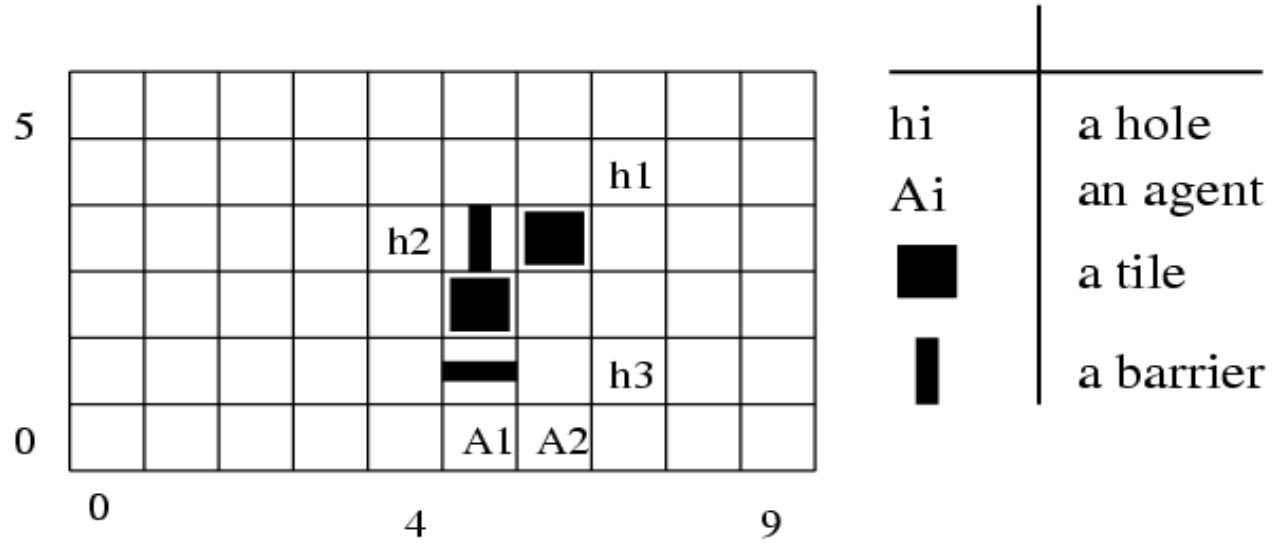


Figure 1: Strongly-Coupled Interactions

- From Goldman and Rosenschein, AAAI-94:

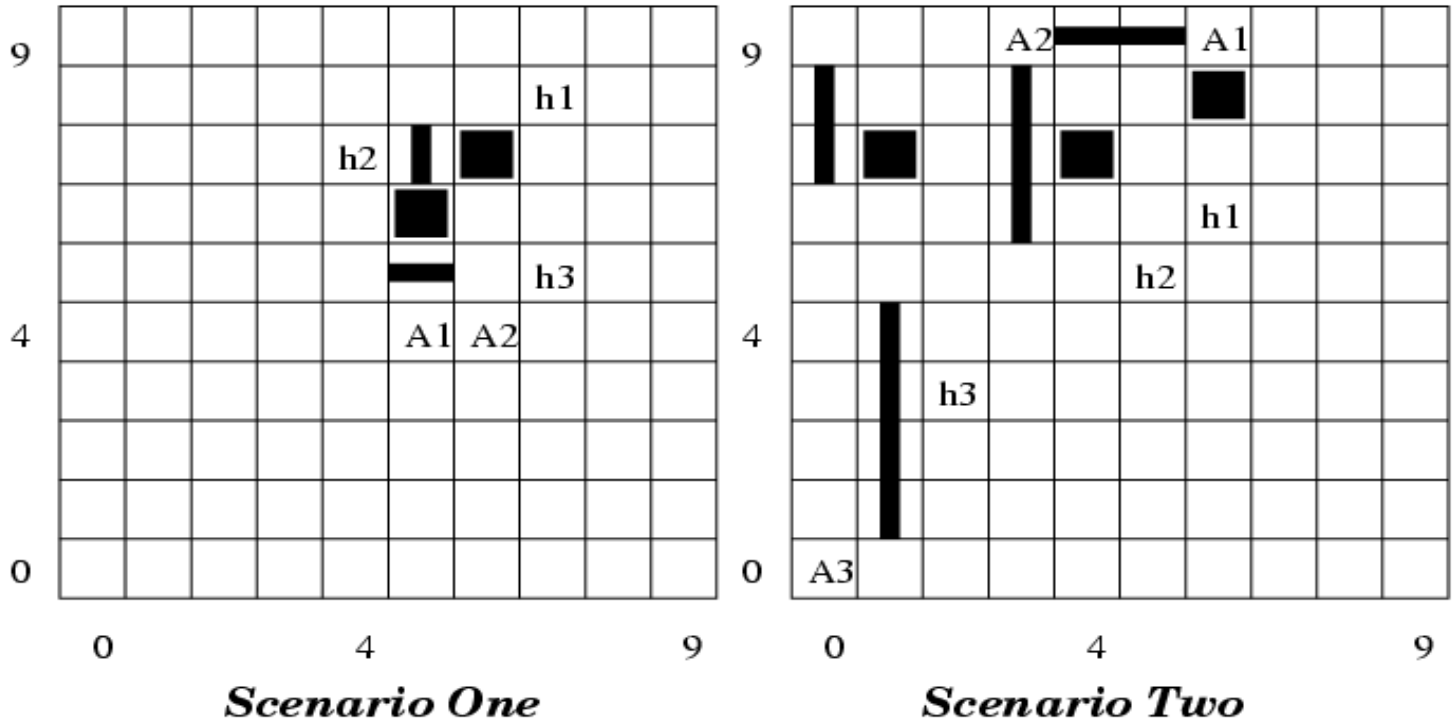


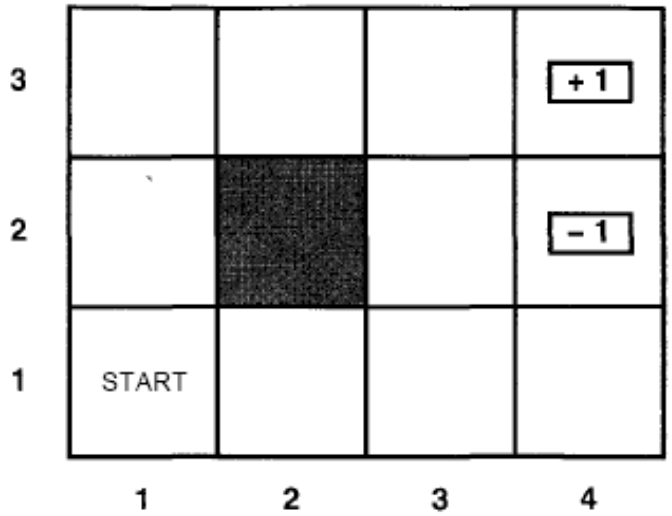
Figure 2: Simulations

- Write $P(r \mid Ag, Env)$ to denote probability that run r occurs when agent Ag is placed in environment Env
  Note: $$\sum_{r \in \mathcal{R}(Ag, Env)} P(r \mid Ag, Env) = 1.$$
- Then optimal agent Ag_{opt} in an environment Env is the one that *maximizes expected utility*:

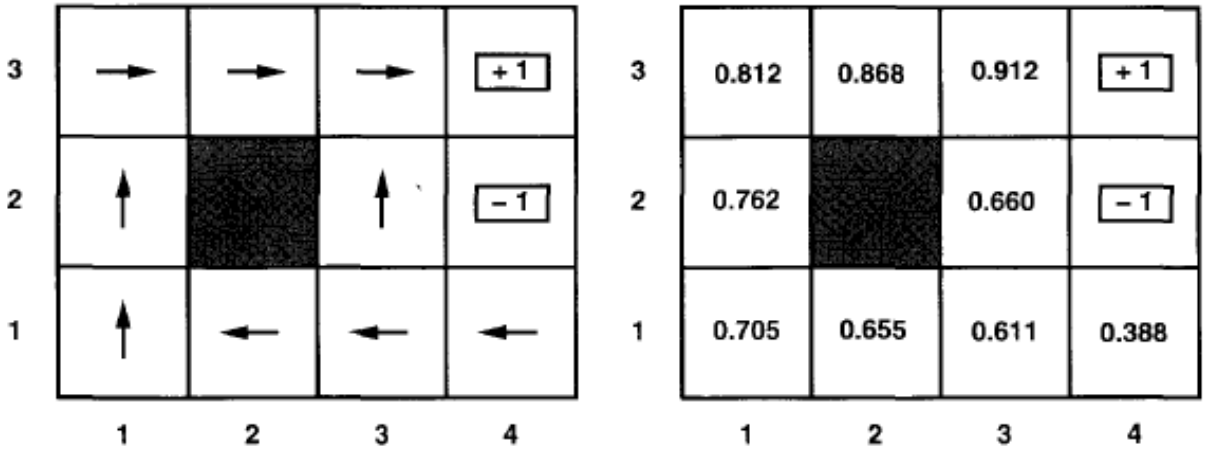
$$Ag_{opt} = \arg \max_{Ag \in \mathcal{AG}} \sum_{r \in \mathcal{R}(Ag, Env)} u(r) P(r \mid Ag, Env). \quad (1)$$

Utility-based Agents

- Not only goals, but utility
- Function that maps states into numbers: usefulness
- Actions maximizing the expected utility



Utility-based Agent



function DT-AGENT(*percept*)**returns** an *action*
static: a set probabilistic beliefs about the state of the world

calculate updated probabilities for current state based on
available evidence including current percept and previous action
calculate outcome probabilities for actions,
given action descriptions and probabilities of current states
select *action* with highest expected utility
given probabilities of outcomes and utility information
return *action*